Dress

A Performance Evaluation of The Intel iAPX 432

Paul M. Hansen, Mark A. Linton, Robert N. Mayo, Marguerite Murphy, and David A. Patterson

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, California 94720
May 17, 1982

ABSTRACT

We describe an experiment to test the 432 as a high-level language uniprocessor by comparing it with the 8086, 68000, and VAX-11/780 for four integer and character programs written in Ada, C, and Pascal.

Introduction

In 1981 Intel announced a 32-bit VLSI microprocessor incorporating several innovations [Intel 81]:

> "The Intel iAPX 432 represents a dramatic advance in computer architecture: it is the first computer whose architecture supports true software-transparent, multiprocessor operation; it is the first commercial system to support an object-oriented programming methodology; it is designed to be programmed entirely in high-level languages; it supports a virtual address space of over a trillion bytes; and it supports on the chip itself the proposed IEEE-standard for floating point arithmetic."

This microcomputer system was the result of a very extensive project that started in 1975 whose goals were [Mazor 81]:

- large scale computational power
- incremental performance capacity
- highly dependable hardware and software
- increased programmer productivity

An interesting question is how much performance is degraded because of the object-oriented architecture with software-transparent multiprocessing and fault tolerance. This report presents the results of a 432 performance study conducted by members of a graduate class at the University of California, Berkeley during the Fall quarter of 1981. We limited ourselves to studying uniprocessor performance and did not consider the following goals for the 432: object-oriented programming, floating point, multiple processes, and multiple processors. The 432 is intended to support Ada [Rattner and Lattin 81], so we used benchmark programs written in high-level languages to determine execution time and code size. Note that a high-level language system consists of the compiler and the machine, so we are not measuring just the architecture and hardware implementation. For purposes of comparison, these programs were run in Pascal on an Intel 8086, a Motorola 68000, and a VAX-11/780, and in C on a 68000 and VAX, as well as in Ada on the 432.

THIS MATERIAL MAY BE PROTECTED BY COPYRIGHT LAW. (TITLE 17 U.C. CODE)

Description of Benchmarks

We measured the code size and execution time of four programs. The time allotted to our visit at Intel prevented us from running more. We did measure the size of other programs and found the results consistent with the measurements presented here. The programs are:

string search
: This program searches a 120 character string for a 15 character substring. It is taken from the performance study sponsored by *Electronic Design News* last year [Grappel and Hemmenway 81].

sieve
: This program computes prime numbers and has also been run on several machines [Gilbreath 81].

puzzle
: This program, created by Forest Baskett, is a bin packing program that solves a simple puzzle and has been run on a wide variety of machines.

acker
: This program computes Ackermann's function with arguments 3 and 6. Ackermann's function is a recursive computation requiring more than 170,000 procedure calls. This benchmark is useful in measuring the cost of a procedure call on a particular machine [Wichmann 76].

All the programs were initially written in C, then translated to Pascal and then into Ada. The C programs do **not** use register variables, pointers, or any other of the unusual features of C.

Accessing parameters and global variables is very expensive inside a procedure in release 2 of the 432 and will be much less expensive in release 3. This overhead can be avoided by either changing the compiler or modifying the program source to make local copies of parameters and globals. We ran Ada programs both translated directly from Pascal and modified to avoid expensive accesses. Table entries marked with a "*" refer to modified programs.

These programs have characteristics found in many high-level language programs except for the frequency of procedure calls and returns. Recent studies on several architectures show that 1 out of every 20 instructions executed is a procedure call or return [Clark and Levy 82] [Ditzel and McLellan 82] [McDaniel 82]. "Acker" is certainly on the high side with 1/7 being call or return, but the dynamic mix for the others is on the low side: 1/235 in "puzzle", 1/245 in "string search", and 1/125000 in "sieve".

Description of Machines

Microprocessor data sheets refer to two terms, *clock rate* and *wait states*, that are sometimes misunderstood. Each microinstruction generally corresponds to one or two clock ticks. Wait states refer to the the time, measured in clock ticks, that a processor is idle while waiting for memory. While we can predict the relative performance of models of the same architecture given the clock rate and wait states, we can not predict relative performance of

different architectures given the same information.

The 432 timings were done on a release 2 system running with a 4 MHz clock and 12 wait states. This system is a half-speed prototype of an 8 MHz system. In December Intel plans to ship release 3 of the 432, the first version in which they have attacked the problem of performance. Timings of release 3 were obtained from a simulator. Size was measured from the output of the release 3 Ada compiler. The 432 systems support virtual memory and provide error correcting memory.

The 8086, announced in 1978, was measured on an Intellec MDS III development system using a 5 MHz part with no wait states. The Pascal compiler was version X125, the first Intel Pascal compiler for the 8086.

The 68000, announced in 1979, was measured on three systems.

- Dual Systems Corporation of Berkeley has a single user UNIX† based 68000 system that uses a variant of the MIT C compiler. The Dual 8312 uses an 8 MHz 68000 built on S-100 cards that needs 2 wait states.

- Motorola's first development system, the EXORMACS, uses an 8 MHz 68000 with 4 wait states: 2 for the memory management unit and 2 for the memory system. The Motorola Pascal compiler is version 2.0, distributed April 10, 1982.

- Finally, Motorola has a 16 MHz 68000 on a board with high-speed memory that runs without wait states.

The VAX-11/780, announced in 1978, supports virtual memory and provides error correcting memory. It was measured with three compilers: the VMS Pascal compiler, the UNIX C compiler, and the Berkeley UNIX Pascal compiler. Berkeley Pascal has an option to get greater performance at the expense of code size by expanding some procedures in line. We ran the programs both ways, with the average change being 4% larger to gain 7% in performance. We selected the time and size of the higher performance choice.

These programs assume a standard word size. The 8086 uses 16 bits, the VAX uses 32 bits, and the 68000 and 432 programs were run with both sizes.

Experimenters

Some of us (Linton, Mayo, and Murphy) ran the 16-bit 432/670 experiments at Intel in Aloha, Oregon. Subsequently, Konrad Lai of Intel ran the programs using 32-bit variables for release 2 and then all the programs for release 3 on a simulator. Ackermann's function was calculated for (1,2) and then multipled by the ratio of acker(3,6) to acker(1,2) on the VAX. Dave Trissel of Motorola ran the 68000 Pascal benchmarks. Members of our department ran the rest of the experiments in Berkeley: Robert Henry measured Pascal programs on the Intel 8086 and the VAX-11/780 under VMS, Keith Sklower ran the C 68000 programs, Peter Kessler did UNIX Pascal on the VAX-11/780, and we ran the UNIX C programs on the VAX-11/780 ourselves.

---

†UNIX is a registered trademark of Bell Laboratories.

Measurements

Table 1(a) shows the execution times of the programs as measured on real hardware. Table 1(b) shows the relative performance of each of the machines with respect to VMS Pascal; entries greater than 1 indicate a faster time than on the VAX.

TABLE 1(a): EXECUTION TIME

Machine	Language	word size	Time (milliseconds) search	sieve	puzzle	acker
VAX-11/780	C	32	1.4	250	9400	4600
	Pascal (UNIX)	32	1.6	220	11900	7800
	Pascal (VMS)	32	1.4	259	11530	9850
68000 (8 MHz)	C	32	4.7	740	37100	7800
	Pascal	16	5.3	810	32470	11480
	Pascal	32	5.8	960	32520	12320
68000 (16 MHz)	Pascal	16	1.3	196	9180	2750
	Pascal	32	1.5	246	9200	3080
8086 (5 MHz)	Pascal	16	7.3	764	44000	11100
432 (4 MHz)	Ada	16	35	3200	350,000	260,000
	Ada	16	14.2*	3200	165,000*	260,000
	Ada	32	16.1*	3200	180,000*	260,000

TABLE 1(b): RELATIVE PERFORMANCE

Machine	Language	word size	Ratio to VMS Pascal (>1 => faster) search	sieve	puzzle	acker	avg±sd
VAX-11/780	C	32	1.0	1.0	1.2	2.1	1.3±.4
	Pascal (UNIX)	32	.9	1.2	1.0	1.3	1.1±.2
	Pascal (VMS)	32	1.0	1.0	1.0	1.0	1.0±.0
68000 (8 MHz)	C	32	.3	.4	.3	1.3	.6±.4
	Pascal	16	.27	.32	.36	.86	.5±.2
	Pascal	32	.24	.27	.35	.80	.4±.2
68000 (16 MHz)	Pascal	16	1.1	1.3	1.3	3.6	1.8±1.0
	Pascal	32	.95	1.0	1.3	3.2	1.6±.9
8086 (5 MHz)	Pascal	16	.2	.3	.3	.9	.4±.3
432 (4 MHz)	Ada	16	.04	.08	.03	.04	.05±.02
	Ada	16	.10*	.08	.07*	.04	.07±.02
	Ada	32	.09*	.08	.06*	.04	.07±.02

Table 2(a) shows the number of bytes of object code and constants for each of the programs. Space necessary for libraries and the operating system was **not** counted. Table 2(b) shows the code sizes relative to VMS Pascal; entries less than 1 indicate code that is smaller than on the VAX.

Although the 432 has bit-variable length instructions, it requires more space than either the 68000 in Pascal or the VAX in C. Reasons include the lack of immediates and the inability to refer to a local variable or constant using fewer than 16 bits of address. On the other hand, the 432 programs are 60% of the size of Pascal on the VAX. Nothing simple explains this seeming contradiction; perhaps a VAX Pascal system requires more sophisticated compiler technology than is currently available.

TABLE 2(a): CODE SIZE

Machine	Language	word size	Size (bytes)			
			484	sieve	puzzle	acker
VAX-11	C	32	764	156	2220	152
	Pascal (UNIX)	32	802	512	4336	244
	Pascal (VMS)	32	636	411	2904	340
68000	C	32	578	228	2940	172
	Pascal	16	578	120	1742	124
	Pascal	32	592	126	1862	130
8086	Pascal	16	756	348	2301	311
432 (rel. 3)	Ada	16	612	180	2443	144

TABLE 2(b): RELATIVE CODE SIZE

Machine	Language	word size	Ratio to VMS Pascal (< 1 => smaller)				
			search	sieve	puzzle	acker	avg±sd
VAX-11	C	32	.60	.38	.77	.45	.5±.2
	Pascal (UNIX)	32	.95	1.24	1.49	.72	1.1±.3
	Pascal (VMS)	32	1.0	1.0	1.0	1.0	1.0±.0
68000	C	32	.79	.55	1.01	.50	.7±.2
	Pascal	16	.72	.29	.60	.36	.5±.2
	Pascal	32	.74	.31	.64	.38	.5±.2
8086	Pascal	16	.94	.85	.79	.91	.9±.1
432 (rel. 3)	Ada	16	.76	.44	.84	.42	.6±.2

Normalized Performance

Table 3 shows the normalized execution times of 8 MHz versions of the 432 in Ada and the 68000 and 8086 in Pascal. The 8086 performance is predicted from the 5 MHz measurements assuming 0 wait states, the 432 programs were run with 4 wait states for both versions of the Ada programs for release 2 and also for release 3. The 4 wait state 68000 was measured on the EXORMACS and the 0 wait state times were computed by doubling the times for the 16 MHz 68000. Times for a 0 wait state 432, while not available as we go to press, are expected to be about 25% faster. The average performance relative to the 68000 is show in figure 1.

TABLE 3: NORMALIZED 8 MHz, 16-bit PERFORMANCE

Wait States	Machine	Language	Time (milliseconds)			
			search	sieve	puzzle	acker
4	68000	Pascal	5.3	810	32470	11480
	432 (rel. 2)	Ada	17.5	1600	175,000	130,000
	432 (rel. 2)	Ada	7.1*	1600	82500*	130,000
	432 (rel. 3)	Ada	4.4	978	45700	47800
0	8086	Pascal	4.6	448	27500	6938
	68000	Pascal	2.6	392	18360	5500

1983 Performance

In the first quarter of 1983, Intel plans to deliver 432/800 systems (release 3, 4 wait states). They also plan to deliver 10 MHz 432 chips. To be fair, we will forecast the 1983 performance of the VAX, 68000, and 8086 as well.

DEC is working on a new Pascal compiler that we expect will result in programs 1.2 to 1.4 times faster than the current version. Since next year marks the fifth anniversary of the VAX-11/780, a faster VAX must be on the horizon.

Motorola has announced two new products for the next year. The 68010 handles page faults, is slightly faster (less than 25%), and is scheduled for this summer. The 68020, scheduled for the end of 1983, has 32-bit internal and external busses plus an on-board instruction cache; however, only one of the three measured 68000 systems has memory management. It seems you must add wait states to add memory management to a 68000, but Sun Microsystems Incorporated has an 8 MHz 68000 with memory management and no wait states. Nevertheless, we believe that faster 68000's will require wait states for memory management. Thus next year we expect systems 1.5 to 3 times faster than the EXORMACS development system with 4 wait states.

Intel has also moved ahead with successors the the 5 MHz 8086. 8 MHz parts are commonly available and Intel has announced 10 MHz versions. The 80286, announced this year, includes an 8086 compatibility mode that runs the same programs many times faster. We thus expect the 1983 version of an 8086 system to be at least 3 times faster than the 5 MHz MDS system.

Conclusion

Our experiment was to test the 432 as a high-level language uniprocessor for integer and character programs. Figure 2 summarizes our findings. The bar graph shows both measured performance on real hardware (solid box) and predicted performance (dotted box). A single 4 MHz, release 2 432 is currently about 1/20th of a VAX-11/780, and we expect Ada programs on the 1983 version to run about 1/5th the speed of Pascal programs on the 1983 VAX. For some applications a 432 system consisting of 5 processors may perform as well as a VAX-11/780. We need multiprocessing benchmarks that could assess such systems.

If performance were the only measure of system cost, then software development would always be done in assembly language. Obviously there are other important aspects, thus we should not disregard the potential 432 benefits of programmer productivity, transparent multiprocessing, and data security. One could attempt to build these 432 functions into software around a 8086, 68000, or VAX, but until then we can not compare completely equal systems. Rather than speculate on the performance of such a system, we have instead tried to evaluate the time and space cost of the 432 approach.

Acknowledgments

Intel has been helpful to us in our study, particularly in allowing us to visit Aloha and run the benchmarks on their release 2 system and release 3 simulator. Konrad Lai and Justin Rattner of Intel were especially helpful in explaining the subtleties of the 432 design. We would also like to thank Tony Anderson and Bill Lattin for arranging the visit. Dave Trissel of Motorola deserves special credit for getting the programs to run on the Motorola Pascal systems, and thanks go to Les Crudele for making the arrangements. We would like to thank several people at Berkeley who helped with the measurements: Wayne Graves, Robert Henry, Paul Israel, Peter Kessler, and Keith Sklower. We are also grateful to Doug Clark, Robert Henry, William Kahan, Carlo Sequin, and John Wakerly for suggesting improvements to this paper.

References

[Clark and Levy 82]
Clark, D., and Levy, H., "Measurement and Analysis of Instruction Use in the VAX-11/780," *Ninth Annual Symposium on Computer Architecture*, Austin, Texas, April 26-29, 1982, pp. 9-17.

[Ditzel and McLellan 82]
Ditzel, D., and McLellan, R., "Register Allocation for Free: The C machine Stack Cache," *Symposium on Architectural Support for Programming Languages and Operating Systems*, Palo Alto, California, March 1-3, 1982, pp. 48-56.

[Gilbreath 81]
Gilbreath, J., "A High-Level Language Benchmark," *Byte*, vol. 6, no. 9, September 1981, pp. 180-198.

[Grappel and Hemmenway 81]
Grappel, R. G., and Hemmenway, J. E., "A Tale of Four Microprocessors: Benchmarks Quantify Performance," *Electronic Design News*, April 1, 1981, pp. 179-265.

[Intel 81]
Introduction to the iAPX 432 Architecture, Intel Corporation, Santa Clara, CA, 1981.

[Mazor 81]
Mazor, S., *432 Architecture Workshop Version 2*, Intel Corporation, July 1981.

[McDaniel 82]
McDaniel, G., "An Analysis of a Mesa Instruction Set Using Dynamic Instruction Frequencies," *Symposium on Architectural Support for Programming Languages and Operating Systems*, Palo Alto, California, March 1-3, 1982, pp. 167-176.

[Rattner and Lattin 81]
Rattner, J., and Lattin, W., "Ada Determines Architecture of 32-bit Microprocessor," *Electronics*, vol. 54, no. 4., February 24, 1981, pp. 119-126.

[Wichmann 76]
Wichmann, B. A., "Ackermann's Function: A Study in the Efficiency of Calling Procedures," *BIT*, vol. 16, 1976, pp. 103-110.

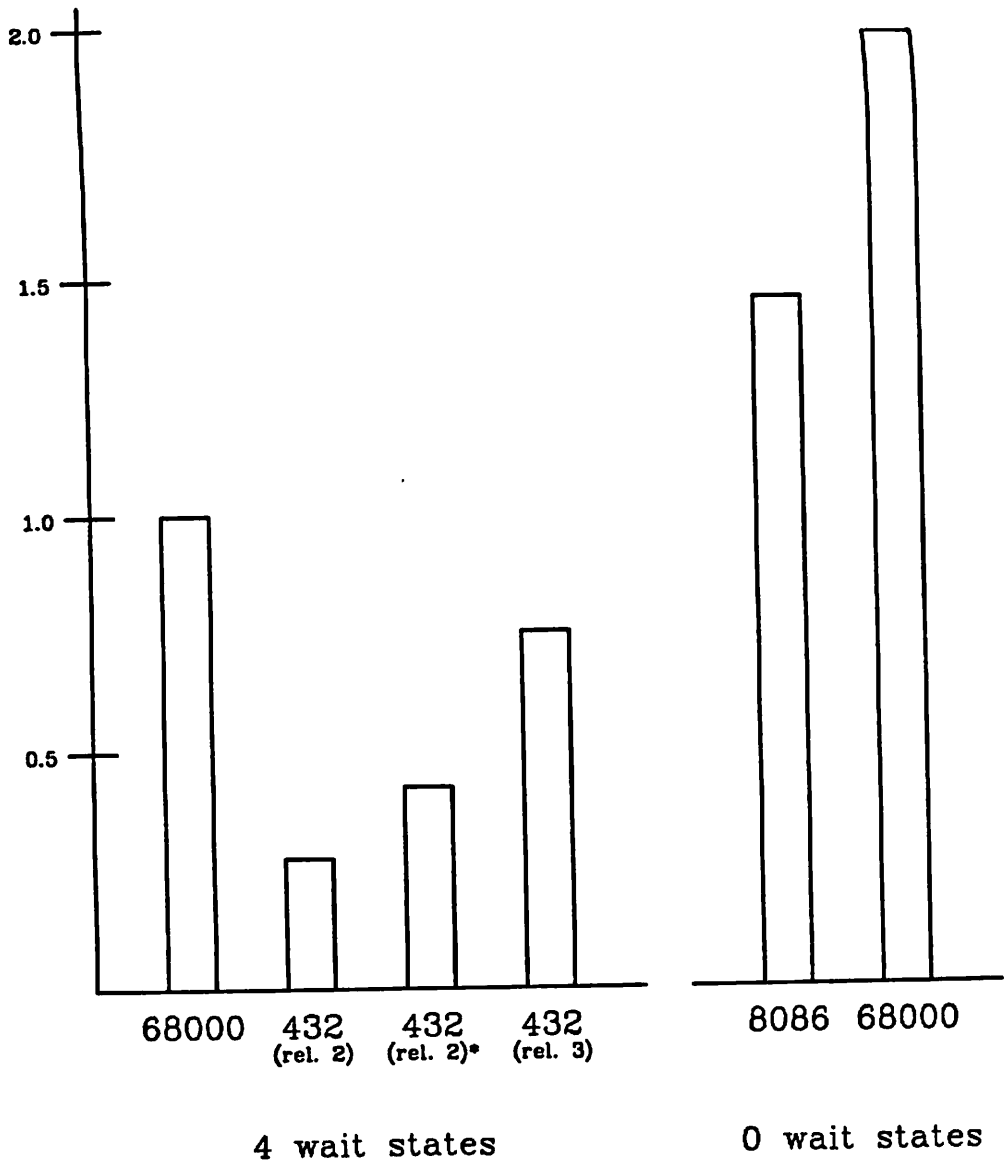

Figure 1: Predicted 8 MHz performance

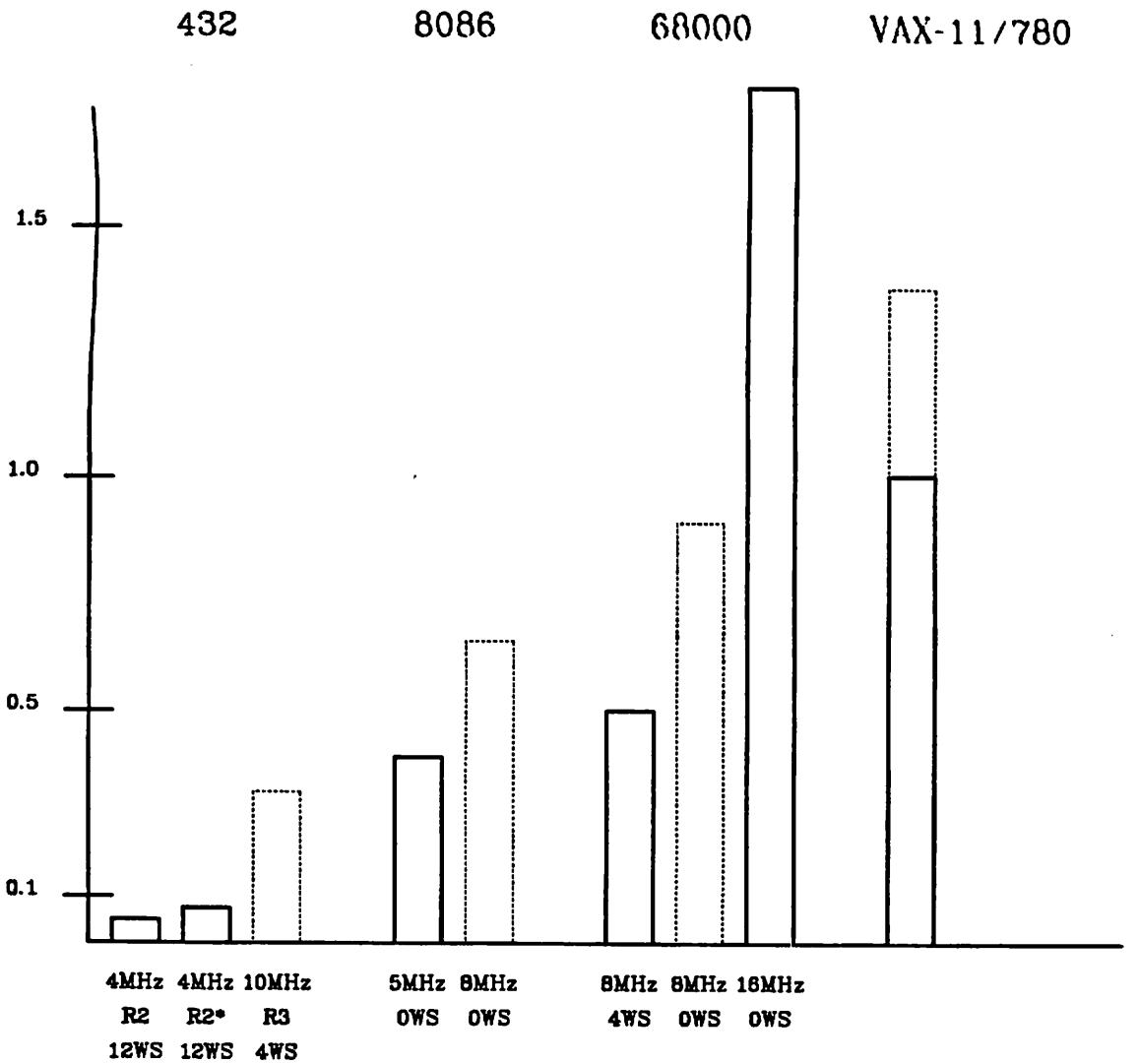

Figure 2: measured and predicted performance